Application Notes
on the
Atari Computer System Interface (ACSI)

The Atari Corporation
Sunnyvale, California
27 September 1985

CONFIDENTIAL

Table of Contents

THE SCOPE OF THIS DOCUMENT is limited to a set of rough application notes on the Atari Computer System Interface. This is a preliminary document and is subject to change without notice.

1. ACSI Bus

- o control signals and a bidirectional bus.
- o target does not receive a command and hold it pending controller ready -- an immediate DEVICE NOT READY error must be sent or the initiator will time out and assume controller nonexistent.
- o controller self test -- recalibrate, ram check, rom checksum, etc.
- o self test always performed following reset -- eliminates need for self test command.
- o initiator could time out (duration to be determined) on a command and reset the target.
- o once the status byte is returned the bus is free.
- o maximum eight bus ports.
- o data transfer rate is up to 8 Mbit/sec.

```
----- ACSI Bus Topology ---------------

 ----------
¦Initiator ¦-------- -------------- -------- . . . --------
¦          ¦       ¦                   ¦                   ¦
 ----------        ¦                   ¦                   ¦
               --------            --------            --------
              ¦Target 0¦          ¦Target 1¦          ¦Target 7¦
               --------            --------            --------
                   ¦                   ¦                ¦   ¦
               -------             -------           ---     ---
              ¦Device ¦           ¦Device ¦         ¦          ¦
               -------             -------        -------    --------
                                                 ¦Device ¦  ¦Device
                                                  -------    --------
```

----- Control and Data Signals --------------

Mnemonic	Name	Characteristics
_RST	Reset	TTL levels, active low.
A1	Address 1	TTL levels.
_IRQ	Interrupt Request	TTL levels, active low, 1 Kohm pullup on initiator side.
_CS	Chip Select	TTL levels, active low.
R/_W	Read/Write	TTL levels.
_DRQ	Data Request	TTL levels, active low, 1 Kohm pullup on initiator side.
_ACK	Acknowledge	TTL levels, active low.
DATA	Data Bus (0-7)	TTL levels.

----- Initiator ACSI Port Pin Assignments --------------

```
INITIATOR    DB 19S                                    TARGET
             ----                                      ----
              1 |<--- Data 0 ---------------------->|
              2 |<--- Data 1 ---------------------->|
              3 |<--- Data 2 ---------------------->|
              4 |<--- Data 3 ---------------------->|
              5 |<--- Data 4 ---------------------->|
              6 |<--- Data 5 ---------------------->|
              7 |<--- Data 6 ---------------------->|
              8 |<--- Data 7 ---------------------->|
              9 |---- Chip Select ----------------->|
             10 |<--- Interrupt Request ------------|
             11 |---- Ground -----------------------|
             12 |---- Reset ----------------------->|
             13 |---- Ground -----------------------|
             14 |---- Acknowledge ----------------->|
             15 |---- Ground -----------------------|
             16 |---- A1 -------------------------->|
             17 |---- Ground -----------------------|
             18 |---- Read/Write ------------------>|
             19 |<--- Data Request -----------------|
             ----                                      ----
```

CONFIDENTIAL

2. ACSI Compliance

2.1. Level 1

- o target will speak only when spoken to.
- o listen to bus during idle -- no disconnect.
- o abort initiator via interrupt.
- o abort target via reset -- software reset must be provided in initiator.
- o RESET HOLD TIME is 12 microseconds.
- o reset has highest bus priority.
- o reset cannot be asserted by a target whether active or inactive.
- o 100 milliseconds before initiator times out on target acknowledgement.
- o CAVEAT: if an initiator prematurely issues a command while the target is executing a command, then the results are unpredictable.
- o device driver in initiator will wait until status byte is returned -- otherwise time out (TBD) and reset target.
- o after receipt of command byte, transaction belongs to controller.
- o target has complete control of bus until status byte is returned.
- o each target should have a user select controller number.

CONFIDENTIAL

HARDWARE.

----- Command Phase ---------------

Data direction: FROM initiator TO target.

```
        .______                              __________________________
A1      .      \____________________________/

        .____________________________            ______________
IRQ     .                            \__________/              \_______

        .____________           ________________           ____________
_CS     .            \_________/                \_________/
                      :        :                 :        :
                      :        :                 :        :
        .______                     ____                      ______
R/_W    .      \_____________________/    \_____________________/
                :     :        :     :     :     :        :     :
                :     :        :     :     :     :        :     :
DATA    =======><-------VALID-------><====><-------VALID-------><====
                :     :        :     :     :     :        :     :
                :<-a->:<--b--->:<-c->:     :<-a->:<--b--->:<-c->:

                 Byte 0                       Byte 1
```

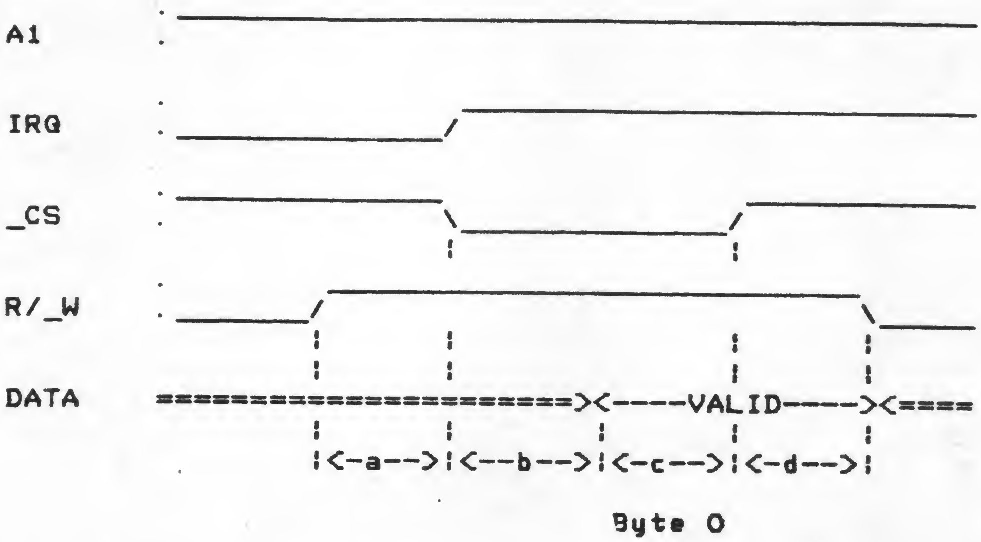
Timing

a) 60 ns (max)
b) 250 ns (max)
c) 20 ns (max)

CONFIDENTIAL

----- Status Phase ---------------

Data direction: FROM target TO initiator.



Timing
```
        a)  50 ns (max)
        b)  150 ns (max)
        c)  100 ns (max)
        d)  80 ns (max)
```

SOFTWARE.

----- Controller Select Byte ---------------

```
        Byte 0  |xxx-----|
                 |||
                  ----------- Controller Number
```

----- Completion Status Byte ---------------

```
        Byte 0  |--------|
```

2.2. Level 2

- include Level 1.
- TEST UNIT READY command is used as a poll.
- NO ERROR is to be interpreted as controller ready.
- DEVICE NOT READY is to be interpreted as controller not ready.

SOFTWARE.

```
----- Command Descriptor Byte ---------------

        Byte 0  |xxxxxxxx|
                 ||||||||
                 |||  -------- Operation Code
                  ----------- Controller Number
```

----- Command Summary Table ---------------

OpCode	Command
0x00	Test Unit Ready

```
----- Completion Status Byte ---------------

        Byte 0  |xxxxxxxx|
                 ||||||||
                 |||  -------- Error Code
                  ----------- Device Number
```

Device Errors

0x00	No Error
0x04	Device Not Ready

Miscellaneous Errors

0x30	Controller Self Test Failed

2.3. Level 3

- include Level 1 and Level 2.

HARDWARE.

----- Data Out Phase ---------------

Data direction: FROM initiator TO target.

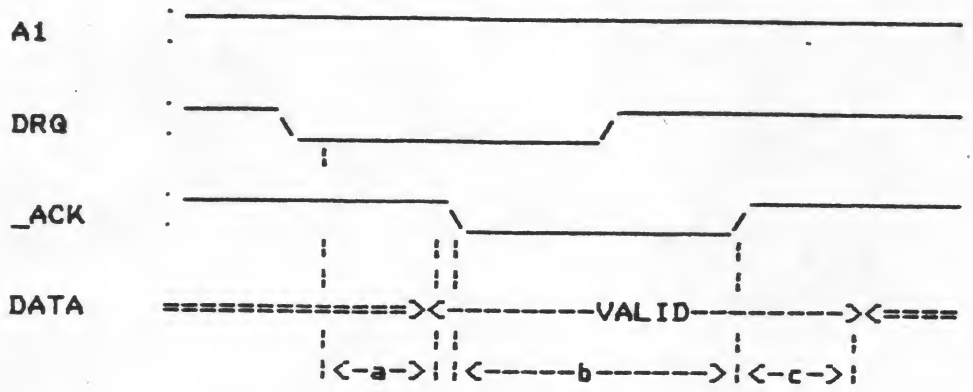
```
        :__________________________________________
A1      :

        :______                  ______________
DRQ     :      \________________/              \___
                 :
        :______________                 ___________
_ACK    :              \_______________/
                 :     :               :
                 :     :               :
DATA    ========><----------VALID------------><====
                 :     :               :     :
                 |<-a->|<------b------>|<-a->|

               |<--c-->|               |<--d-->|
```

Timing

a) 60 ns (max)
b) 250 ns (max)
c) 240 ns (max)
d) 240 ns (min)

----- Data In Phase ---------------

Data direction: FROM target TO initiator.



Timing

a) 60 ns (max)
b) 250 ns (max)
c) 50 ns (min)

SOFTWARE.

```
----- ACSI Command Descriptor Block ---------------

        Byte 0  ¦xxxxxxxx¦
                 ¦¦¦¦¦¦¦¦
                 ¦¦¦ -------- Operation Code
                  ----------- Controller Number
        Byte 1  ¦xxxxxxxx¦
                 ¦¦¦¦¦¦¦¦
                 ¦¦¦ -------- Block Address High
                  ----------- Device Number
        Byte 2  ¦xxxxxxxx¦
                 ¦¦¦¦¦¦¦¦
                  ----------- Block Address Mid
        Byte 3  ¦xxxxxxxx¦
                 ¦¦¦¦¦¦¦¦
                  ----------- Block Address Low
        Byte 4  ¦xxxxxxxx¦
                 ¦¦¦¦¦¦¦¦
                  ----------- Block Count
        Byte 5  ¦xxxxxxxx¦
                 ¦¦¦¦¦¦¦¦
                  ----------- Control Byte
```

----- Command Summary Table ---------------

OpCode	Command	
0x00	Test Unit Ready	
0x08	Read	*
0x0a	Write	*
0x0b	Seek	
0x1a	Mode Sense	

* multisector transfer with implied seek

Command Errors

0x20	Invalid Command
0x21	Invalid Address
0x23	Volume Overflow
0x24	Invalid Argument
0x25	Invalid Device Number

3. ACSI Initiator

- must transfer data in 16 byte increment blocks.
- must use ST BIOS system variable flock (see A Hitchhiker's Guide to the BIOS).

----- Initiator Handshake Sequence ---------------

- load DMA Base Address Register.
- toggle Write/_Read to clear status (DMA Mode Control Register)
- select DMA read or write (DMA Mode Control Register).
- select DMA Sector Count Register (DMA Mode Control Register).
- load DMA Sector Count Register (DMA operation trigger).
- select controller internal command register (DMA Mode Control Register).
- issue controller select byte by clearing A0 to 0.
- set A0 to 1 for remaining command bytes.
- after last command byte select controller (DMA Mode Control Register).
- DMA active until sector count is zero (DMA Status Register, do not poll during DMA active).
- check DMA error status (DMA Status Register).
- check controller status byte.

```
loadable        equ     1                       ; nonzero for loadable driver

*-----------------------------------------------------------------------------
*                                                                           :
*       ST SASI Hard disk driver                                            :
*       (C)1985 Atari Corp.                                                 :
*                                                                           :
*----                                                                       :
*  9-Apr-1985 lmd       Hacked it up.  "Gee, it seems to work ..."          :
* 14-Apr-1985 lmd       linked with BIOS (***FOR NOW***)                    :
* 20-Apr-1985 lmd       hacked for WD controller (now, wired...)            :
* 24-Jun-1985 jwt       hacked for Adaptec, new kludge board                :
* 01-Jul-1985 jwt       seems to work, add more formatting and more         :
*                        detailed error reporting                           :
* 22-Jul-1985 jwt       change timing of wdc/wdl at start of command,       :
*                        added extra move.w $8a,wdl to change A1            :
* 23-Jul-1985 jwt       use a move.l instruction for all wdc/wdl write      :
*                        pairs since it changes A1 quickly enough that      :
*                        the (old) DMA chip does not incorrectly            :
*                        generate two chip selects                          :
*                                                                           :
*-----------------------------------------------------------------------------


flock           equ     $43e                    ; FIFO lock variable
hdv_init        equ     $46a                    ; hdv_init()
hdv_bpb         equ     $472                    ; hdv_bpb(dev)
hdv_rw          equ     $476                    ; hdv_rw(rw, buf, count, recno, dev)
hdv_boot        equ     $47a                    ; hdv_boot()
hdv_mediach     equ     $47e                    ; hdv_mediach(dev)
_drvbits        equ     $4c2                    ; block device bitVector
_dskbufp        equ     $4c6                    ; pointer to common disk buffer

nretries        equ     3                       ; #retries-1


* ---------------- Installer ----------------
        .globl  i_sasi
i_sasi: bra     i_sasi2

        dc.b    '@(#)ahdx v0.04',$0d,$0a,0,$1A



* ---------------- Front End ----------------


*+
* LONG hbpb(dev) - return ptr to BPB (or NULL)
*
* Passed:       dev     4(sp).W
*
*-
hbpb:
        move.w  4(sp),d0                        ; d0 = devno
        move.l  o_bpb,a0                        ; a0 -> pass-through vector
```

CONFIDENTIAL

```
        lea     _sasi_bpb(pc),a1            ; a1 -> our handler
        bra     check_dev                   ; do it


*+
* LONG rw(rw, buf, count, recno, dev)
*
* Passed:       dev     $e(sp).W
*               recno   $c(sp).W
*               count   $a(sp).W
*               buf     6(sp).L
*               rw      4(sp).W
*
*-
hrw:
        move.w  $e(sp),d0                   ; d0 = devno
        move.l  o_rw,a0                     ; a0 -> pass-through vector
        lea     _sasi_rw(pc),a1             ; a1 -> our handler
        bra     check_dev                   ; do it


*+
* LONG mediach(dev)
*
* Passed:       dev     4(sp).W
*
*-
hmediach:
        move.w  4(sp),d0                    ; d0 = devno
        move.l  o_mediach,a0                ; a0 -> pass-through vector
        lea     _sasi_mediach(pc),a1        ; a1 -> our handler


*+
* check_dev - use handler, or pass vector through
*
* Passed:       d0.w = device#
*               a0 ->  old handler
*               a1 ->  new handler
*               a5 ->  $0000 (zero-page ptr)
*
* Jumps-to:     (a1) if dev in range for this handler
*               (a0) otherwise
*
*-
check_dev:
        cmp.w   #2,d0                       ; devnos match?
        bne     chkd_f                      ; (no)
        move.l  a1,a0                       ; yes -- follow success vector
chkd_f: jmp     (a0)                        ; do it


* --------------- Medium level driver ---------------
```

```
*+
* _sasi_init - initialize SASI dev
* Passed:        nothing
* Returns:       d0 < 0: error
*                d0 ==0: success
* function performed by _hinit.... and the assembler won't
*  let me have a forward reference here
*-
*       .globl  _sasi_init
*_sasi_init: equ        _hinit

*+
* _sasi_bpb - return BPB for hard drive
* Synopsis:      LONG _sasi_bpb(dev)
*                WORD dev;
*
* Returns:       NULL, or a pointer to the BPB buffer
*
*-
        .globl  _sasi_bpb
_sasi_bpb:
        move.l  #thebpb,d0
        rts


*+
* _sasi_rw - read/write hard sectors
* Synopsis:      _sasi_rw(rw, buf, count, recno, dev)
*
* Passed:        dev     $e(sp).W
*                recno   $c(sp).W
*                count   $a(sp).W
*                buf     6(sp).L
*                rw      4(sp).W          ; non-zero -> write
*
*-
        .globl  _sasi_rw
_sasi_rw:
        move.w  #nretries,retrycnt       ; setup retry counter

sasrw1: moveq   #0,d0                    ; coerce word to long, unsigned
        move.w  $c(sp),d0                ; sect.L
        move.w  $a(sp),d1                ; count.W
        move.l  6(sp),d2                 ; buf.L
        move.w  4(sp),d3                 ; rw

        clr.w   -(sp)                    ; dev = 0
        move.l  d2,-(sp)                 ; buf
        move.w  d1,-(sp)                 ; count
        move.l  d0,-(sp)                 ; sect
        tst.w   d3                       ; read or write?
        bne     sasrw3                   ; (write)
        bsr     _hread                   ; read sectors
        bra     sasrw2
sasrw3: bsr     _hwrite                  ; write sectors
```

CONFIDENTIAL

```
sasrw2: add.w   #12,sp                  ; (cleanup stack)
        tst.l   d0                      ; errors?
        beq     sasrwr                  ; no -- success
        subq.w  #1,retrycnt             ; drop retry count and retry
        bpl     sasrw1

sasrwr: rts


*+
* _sasi_mediach - see if hard disk media has changed (it never does)
* Synopsis:     _sasi_mediach(dev)
*               WORD dev;
*
* Returns:      0L
*
*-
        .globl  _sasi_mediach
_sasi_mediach:
        clr.l   d0
        rts


*+
* BPB for 10MB drive
* Approximate only.  Tweak me.
*
*-
thebpb: dc.w    512                     ; #bytes/sector
        dc.w    2                       ; #sectors/cluster
        dc.w    1024                    ; #bytes/cluster
        dc.w    16                      ; rdlen (256 root files) (in sector
        dc.w    41                      ; FATsiz (10300 FAT entries) (secto
        dc.w    42                      ; 2nd FAT start
        dc.w    99                      ; data start (in sectors)
        dc.w    10300                   ; #clusters (approximate here)
        dc.w    1                       ; flags (16-bit FATs)


* ---------------- Low-level driver ----------------


*----- Globals
flock           equ     $43e            ; FIFO lock variable
_hz_200         equ     $4ba            ; 200hz system ticker


*----- Hardware:
wdc             equ     $ff8604
wdl             equ     $ff8606
wdcwdl          equ     wdc             ; used for long writes
dmahi           equ     $ff8609
dmamid          equ     dmahi+2
```

```
malow           equ     dmamid+2
pip             equ     $fffa01


------ Tunable:
timeout                     equ     $80000          ; long-timeout
timeout                     equ     $80000          ; short-timeout


*+
* LONG _qdone() - Wait for operation complete
* Passed:       nothing
*
* Returns:      EQ: no timeout
*               MI: timeout condition
*
* Uses:         D0
*-
_qdone:
        move.l  #ltimeout,tocount
qd1:    subq.l  #1,tocount                  ; drop timeout count
        bmi     qdq                         ; (i give up, return NE)
        move.b  gpip,d0                     ; interrupt?
        and.b   #$20,d0
        bne     qd1                         ; (not yet)

        moveq   #0,d0                       ; return EQ (no timeout)
qdq:    rts


*+
* WORD _endcmd()
* Wait for end of SASI command
* Passed:       d0 value to be written to wdl
*
* Returns:      EQ: success (error code in D0.W)
*               MI: timeout
*               NE: failure (SASI error code in D0.W)
*
* Uses:         d0,d1
*-
_endcmd: move   d0,d1                       ; preserve wdl value

        bsr     _qdone                      ; wait for operation complete
        bmi     endce                       ; (timed-out, so complain)

        move.w  d1,wdl
        nop
        move.w  wdc,d0                      ; get the result
        and.w   #$00ff,d0                   ; (clean it up), if non-zero should
endce:  rts                                 ;  do a ReadSense command to learn more


*+
* _hinit(dev)
```

CONFIDENTIAL

```
* WORD dev;
* Initialize hard disk
*
* Returns:        -1 if hard disk not there
*
*-
        .globl  _sasi_init
_sasi_init:
_hinit:
        pea     actur                   ; push test unit read command block
        bsr     _dosahdxc
        addq.l  #4,sp
        rts

*-
* _hread(sectno, count, buf, dev)
* LONG sectno;          4(sp)
* WORD count;           8(sp)
* LONG buf;            $a(sp)  $b=high, $c=mid, $d=low
* WORD dev;            $e(sp)
*
* Returns:        -1 on timeout
*                 0 on success
*                 nonzero on error
*
*-
        .globl  _hread
_hread:
        st      flock                   ; lock FIFO

        move    #$88,wdl
        move.l  #$0008008a,wdcwdl       ; 08 wdc, 8a wdl

        move.l  $a(sp),-(sp)            ; set DMA address
        bsr     _setdma
        addq    #4,sp

        bsr     _setss                  ; set sector and size
        bmi     _hto

        move.w  #$190,wdl
        nop
        move.w  #$90,wdl
        nop
        move.w  8(sp),wdc               ; write sector count to DMA chip
        nop
        move.w  #$8a,wdl
        nop
        move.l  #$00000000,wdcwdl       ; control byte  0 wdc 0 wdl

        move.w  #$8a,d0
        bsr     _endcmd

hrx:    bra     _hdone                  ; cleanup after IRQ
```

```
*-
* _hwrite(sectno, count, buf, dev)
* LONG sectno;            4(sp)
* WORD count;             8(sp)
* LONG buf;              $a(sp)  $b=high, $c=mid, $d=low
* WORD dev;              $e(sp)
*
*-
        .globl  _hwrite
_hwrite:
        st      flock                   ; lock FIFO

        move.l  $a(sp),-(sp)            ; set DMA address
        bsr     _setdma
        addq    #4,sp

        move.w  #$88,wdl
        move.l  #$000a008a,wdcwdl       ; 0a wdc 8a wdl

        bsr     _setss
        bmi     _hto

        move.w  #$90,wdl
        nop
        move.w  #$190,wdl
        nop
        move.w  8(sp),wdc               ; sector count for DMA ch
        nop
        move.w  #$18a,wdl
        nop
        move.l  #$00000100,wdcwdl

        move.w  #$18a,d0
        bsr     _endcmd

hwx:    bra     _hdone                  ; cleanup after IRQ


*+
* _wd_format - format WD hard disk
* Passed:       nothing
* Returns:      0, or -N
* Uses:         <..?..>
*
*-
        .globl  _wd_format
_wd_format: lea acfmt,a0                ; pick up pointer to the
        clr.w   d0
        st      flock                   ; lock FIFO
        move.w  #$88,wdl
        move.b  (a0)+,d0                ; get the command byte
        swap    d0
        move.w  #$8a,d0
        move.l  d0,wdc                  ; byte wdc 8a wdl

        moveq   #(5-1),d1               ; write remaining 5 byte
```

CONFIDENTIAL

```
nt1:    bsr     _qdone                  ;  (presumes only one unit)
        bmi     _hto
        move.b  (a0)+,d0                ; next byte of command
        swap    d0
        move.w  #$8a,d0
        move.l  d0,wdcwdl
        dbra    d1,fmt1

nt2:    btst    #5,gpip                 ; wait (forever) for completion
        bne     fmt2

        move.w  wdc,d0                  ; get the status
        andi.w  #$00FF,d0               ; only low byte is significant

        bra     _hdone                  ; cleanup after IRQ

+
 _wd_setup - setup parameters for WD hard disk

-
        .globl  _wd_setup
wd_setup:
        st      flock
        pea     adap_parms(pc)
        bsr     _setdma
        addq    #4,sp

        move.w  #$88,wdl
        move.l  #$0015008a,wdcwdl       ; mode select command  15 wdc 8a wdl

        bsr     _qdone
        bmi     wdx
        move.l  #$0000008a,wdcwdl
        bsr     _qdone
        bmi     wdx
        move.l  #$0000008a,wdcwdl
        bsr     _qdone
        bmi     wdx
        move.l  #$0000008a,wdcwdl
        bsr     _qdone
        bmi     wdx
        move.l  #$0016008a,wdcwdl       ; 22 bytes of parameters

        bsr     _qdone
        bmi     wdx
        move.w  #$90,wdl                ; reset the DMA chip
        nop
        move.w  #$190,wdl
        nop
        move.w  #$01,wdc                ; 1 sector of DMA (actually less)
        nop
        move.w  #$18a,wdl
        nop
        move.l  #$00000100,wdcwdl       ; control byte

        move.w  #$18a,d0                ; wdl value
        bsr     _endcmd
lx:     bra     _hdone

--- parameters for 10MB WD
lap_parms:  dc.b $00, $00, $00, $08, $00, $00, $00, $00, $00, $00
        dc.b    $02, $00, $01, $02, $62, $02, $01, $00, $01, $00, $00, $02


+
 LONG _dosahdxc( addr ) BYTE *addr;
        do a simple (no DMA) ahdx command
-
        .globl  _dosahdxc
dosahdxc: movea.l 4(sp),a0                  ; pick up pointer to the command block
        clr.w   d0
        st      flock                       ; lock FIFO
        move.w  #$88,wdl
        move.b  (a0)+,d0                    ; get the command byte
        swap    d0
        move.w  #$8a,d0
        move.l  d0,wdcwdl                   ; send it to the controller

        moveq   #(5-1),d1                   ; write remaining 5 bytes of command
osac1:  bsr     _qdone                      ;  (presumes only one unit)
        bmi     _hto
        move.b  (a0)+,d0                    ; next byte of command
        swap    d0
        move.w  #$8a,d0
        move.l  d0,wdcwdl
        dbra    d1,dosac1

        bsr     _qdone                      ; wait for the command to complete
        bmi     _hto

        move.w  wdc,d0                      ; get the status
        andi.w  #$00FF,d0                   ; only low byte is significant

        bra     _hdone                      ; cleanup after IRQ


*+
* void _setdma(addr)
* LONG addr;
*-
_setdma:
        move.b  7(sp),dmalow
        move.b  6(sp),dmamid
        move.b  5(sp),dmahi
        rts

*+
* WORD _setss  -- set sector number and number of sectors
*-

_setss: move.w  #$8a,wdl
```

CONFIDENTIAL

```
        bsr     _qdone                  ; wait for controller to take comman
        bmi     setsse

        move.b  9(sp),d0                ; construct sector#
        move.b  $e(sp),d1               ; ORed with devno
        lsl.b   #5,d1
        or.b    d1,d0
        swap    d0
        move.w  #$008a,d0
        move.l  d0,wdcwdl               ; write MSB sector# + devno
        bsr     _qdone
        bmi     setsse

        move.b  10(sp),d0               ; write MidSB sector#
        swap    d0
        move.w  #$008a,d0
        move.l  d0,wdcwdl
        bsr     _qdone
        bmi     setsse

        move.b  11(sp),d0               ; write LSB sector#
        swap    d0
        move.w  #$008a,d0
        move.l  d0,wdcwdl
        bsr     _qdone
        bmi     setsse

        move.w  12(sp),d0               ; write sector count
        swap    d0
        move.w  #$008a,d0
        move.l  d0,wdcwdl
        bsr     _qdone

setsse: rts

_hto:   moveq   #-1,d0                  ; indicate timeout
_hdone: move.w  #$80,wdl                ; Landon's code seems to presume we
        nop                             ;  put this back to $80
        tst.w   wdc
        clr     flock                   ; NOW, signal that we are done
        rts

savssp:         dc.l    1               ; (saved SSP)
tocount:        dc.l    1               ; timeout counter
retrycnt:       dc.w    1               ; retry counter
o_init:         dc.l    1
o_bpb:          dc.l    1
o_rw:           dc.l    1
o_mediach:      dc.l    1

i_sasi2: nop

 ifne loadable
        clr.l   -(sp)                   ; it's a bird...
        move.w  #$20,-(sp)              ;    ... it's a plane ...
        trap    #1                      ;      ... no, its:
```

CONFIDENTIAL

```
        addq    #6,sp                   ; SOOUPERUSER!
        move.l  d0,savssp               ; "Faster than a prefetched opcode..."
 endc

        bsr     _sasi_init              ; kick controller
        tst.w   d0
        bne     isase                   ; punt -- disk didn't respond correctly

        clr.l   d0
        or.l    _drvbits,d0             ; include C: bit in devVector
        or.l    #$4,d0
        move.l  d0,_drvbits

        clr.l   a5                      ; zeropage ptr
        move.l  hdv_bpb(a5),o_bpb               ; save old vectors
        move.l  hdv_rw(a5),o_rw
        move.l  hdv_mediach(a5),o_mediach

        move.l  #hbpb,hdv_bpb(a5)               ; install our new ones
        move.l  #hrw,hdv_rw(a5)
        move.l  #hmediach,hdv_mediach(a5)

isasq:  nop                             ; stupid assembler

 ifne loadable
        move.l  savssp,-(sp)            ; become a mild mannered user process
        move.w  #$20,-(sp)
        trap    #1
        addq    #6,sp
 endc

 ifne loadable
        move.w  #0,-(sp)        ; exit code
        move.l  #((i_sasi2-i_sasi)+$0100),-(sp) ; save code, data, & basepage
        move.w  #$31,-(sp)      ; terminate and stay resident
        trap    #1              ; should never come back...
 endc

        rts

isase:  lea     nodmsg,a0
        bsr     msg

 ifne loadable
        move.l  savssp,-(sp)            ; become a mild mannered user process
        move.w  #$20,-(sp)
        trap    #1
        addq    #6,sp
 endc

        move.w  #1,-(sp)                        ; flag error status
        move.w  #$4c,-(sp)              ; terminate
        trap    #1

msg:    move.l  a0,-(sp)
        move.w  #9,-(sp)                ; print null terminated string
        trap    #1
        addq.l  #6,sp
        rts

actur:  dc.b    0,0,0,0,0,0             ; atari command: test unit ready
acfmt:  dc.b    4,0,0,0,1,0             ;                format disk

nodmsg: dc.b    'No AHDX disk response.',$0d,$0a,0

        .even

        end
```